



INTERNATIONALIZATION, LOCALIZATION, and MULTILINGUAL INTERFACES

Dropbox:

<https://www.dropbox.com/sh/r8j884d8jggwomj/dXUgTvbyM>

<http://bit.ly/1kxNdiy>

David Allen

Quantum Mobile Solutions
dallen@qmobilesolutions.com

Developing your Apps so that they can be used by people that speak languages other than English and/or who live in other countries or come from different cultures, can massively increase your audience or market.

Most Apps do not cater for multiple cultures, so there is significant opportunity if your Apps do, giving you an additional unique selling point and niches.

The main reasons for doing so are:

- Access to markets in other countries
- Accessibility to ethnic groups
- A better experience for your users based on their culture and primary language

This can:

- Increase your market size
- Give you an additional Unique Selling Point
- Provide access to low competition niches

main areas for internationalization / localization

Localization is not just a matter of providing you interface in multiple languages. Though this is the most important aspect, there are several other aspects that should be considered:

- Linguistic formatting:
 - Dates e.g. 12/18/13 vs 18/12/13
 - Numbers e.g. 100000 vs 100,000 vs 100.000
 - Alphabet e.g. Latin vs Cyrillic vs Greek vs Arabic etc.
 - Layout order e.g. Left to Right vs Right to Left
- Cultural formatting:
 - Colours: e.g. West: Red = Danger, East: Red = Luck
 - Layout: e.g. Placing emphasis variations
 - Cultural References e.g. Idioms, numbers, animals etc

- Decide on your target **locales** and the granularity of localization
- Get people who are native to the countries, cultures, languages you are targeting to advise on design, layout and language
- Decide which aspects are the most important for your target locales, to reduce the impact and work. This will depend on both the targets and the purpose of your App:
 - Language will almost always be the #1 priority
- Look for 'domain examples' targeted at your target locales:
 - Websites - many allow you to select a language
 - Books
 - Journals
 - Other Apps
- Beware of trusting 'friends translations' & being too colloquial -common mistake
- Always get several 'native language' speakers to review and test your app

translation differences dialects & colloquializations

Differences can exhibit themselves in spelling, grammar, the words used and accepted phrases.

Consider the differences between British English, American English, Australian English (let alone Jamaican English and South African English)!

American

Cookie

Biscuit

Trunk

Eraser

Rubber

Rancher

Squatter

Elevator

Gas

Color

Cantaloupe

British

Biscuit

Savory Scone

Boot

Rubber

Condom

Farmer

Squatter

Lift

Petrol

Colour

Melon

Australian

Biscuit

Savory Scone

Boot

Rubber

Condom or Rubber!

Squatter

Squatter

Lift

Petrol

Colour

Rockmelon

translation differences, dialects & colloquializations

At least the combination of films, television, books and theatre makes both American and British English widely accepted and understood.

But consider Spanish. Many different countries have slightly different versions: Mexican Spanish is different from Porto Rican Spanish is different from Catalan Spanish is different from Castilian Spanish

Then there are colloquialisms. Unless someone is linguistically aware and/or trained, they may not even realize many of the colloquialism they use – this is a huge risk when getting friends to translate things.

Unless you are aiming at a particular cultural sub-group, and specific localization to that group is important, select the 'Gold Standard(s)' you will use for the language that all users will understand and accept:

Spanish: Castilian Spanish

Portuguese: Portugal Portuguese

French: French French

Chinese: Cantonese and Mandarin

English: American and/or British English

android's support for localization

locales - resource files

Resource files contain data that can be access by the Android operating system, the complier, layouts and code in pre defined ways

- Resource files exist in the .res folder of an Android project
- There are reserved directory names under the .res folder which contain different types of data files
- These folders can be 'specialized' so that Android automatically uses the most appropriate one by adding specifics to the directory name:
 - drawable - contains default image resources
 - drawable-hdpi - contains image resources for hdpi displays
- Localization is supported using **locale** codes specified in the format:
 - -xx or -xx-rYY where xx represents the main locale (language) code and YY represents the regional locale code
- **Locale** codes are loosely based on ISO standards ISO 639-1 & 3166-1
- Different versions of Android support different codes
- It's up to the manufacturer to provide the supporting system files



android's support for localization resource files

- Beware, it's possible to create a locale specific resource for a locale that does not exist, is not supported by the version of Android or can not be selected automatically because it is not provided by the manufacturer.
- **Locales** are automatically selected by Android via the language selected by the user
 - Android will automatically use the most specific matching resource files available
 - This can be overridden programmatically

- Examples of **locale** codes:

English: -en	British: en-rGB	American: en-rUS	Philippines: en-rPH
Spanish: -es	Spain: es-rES	Mexican: es-rMX	Puerto Rican: es-rPR
French: -fr	France: fr_rFR	Belgium: fr_rBE	Canada: fr-rCA

- Examples of folders using locale codes:

values-en	values-en-rGB	values-en-rUS	values-en-rPH
values-es	values-es-rES	values-es-rMX	values-es-rPR
drawable-fr	drawable-fr-rFR	drawable-fr-rBE	drawable-fr-rCA



overriding the locale programmatically

Android provides the `Locale` class to specify the locale that should be used

Beware: Android uses `Locale` in various places and usually defaults to the `System Locale` unless overridden for the given instance of the object that uses it. This can lead to unexpected behavior unless careful.

Different versions of Android may use `Locale` slightly differently.

- `Locale` is used by the `Activity` class (via the `Application` class) to fetch the correct locale specific resources, adjust the screen, set the correct number and sentence formatting etc.
- It may be overridden in the `Activity` class by creating a new `Locale` class and assigning it to the `Activity`, BEFORE setting the `ContentView`:

```
Locale locale = new Locale("en-rGB");
Locale.setDefault(locale);
Configuration config = new Configuration();
config.locale = locale;
getBaseContext().getResources().updateConfiguration
( config, getBaseContext().getResources().getDisplayMetrics() );
setContentView(R.layout.screen_name);
```

overriding the locale programmatically

- New instances of Activity always use the default System Locale set by the OS language selection (unless the Activity is cached), thus:
 - You must re-assign the new Locale for every new Activity created
 - You must Assign the new Locale every time an Activity is destroyed and created (i.e. on a configuration change)
 - Best done by creating a custom class that inherits Activity, and overrides `onCreate()` and `onConfigurationChanged(Configuration c)`
- As the locale changes are applied when the Activity is recreated, if you want them to apply to the current screen, you will need to destroy and create the current Activity programmatically (remember save and restore any Activity state data so that the screen looks the same (i.e. selections selected, menus drawn etc))
- Some Manifest settings may also be required for each Activity, depending on the API level being compiled against. Try:
 - `android:configChanges="locale|layoutDirection|....."` - *for each Activity*
 - `<supports-screens android:smallScreens="true"
android:normalScreens="true" android:largeScreens="true"
android:anyDensity="true" />` - *before the Application statement block*

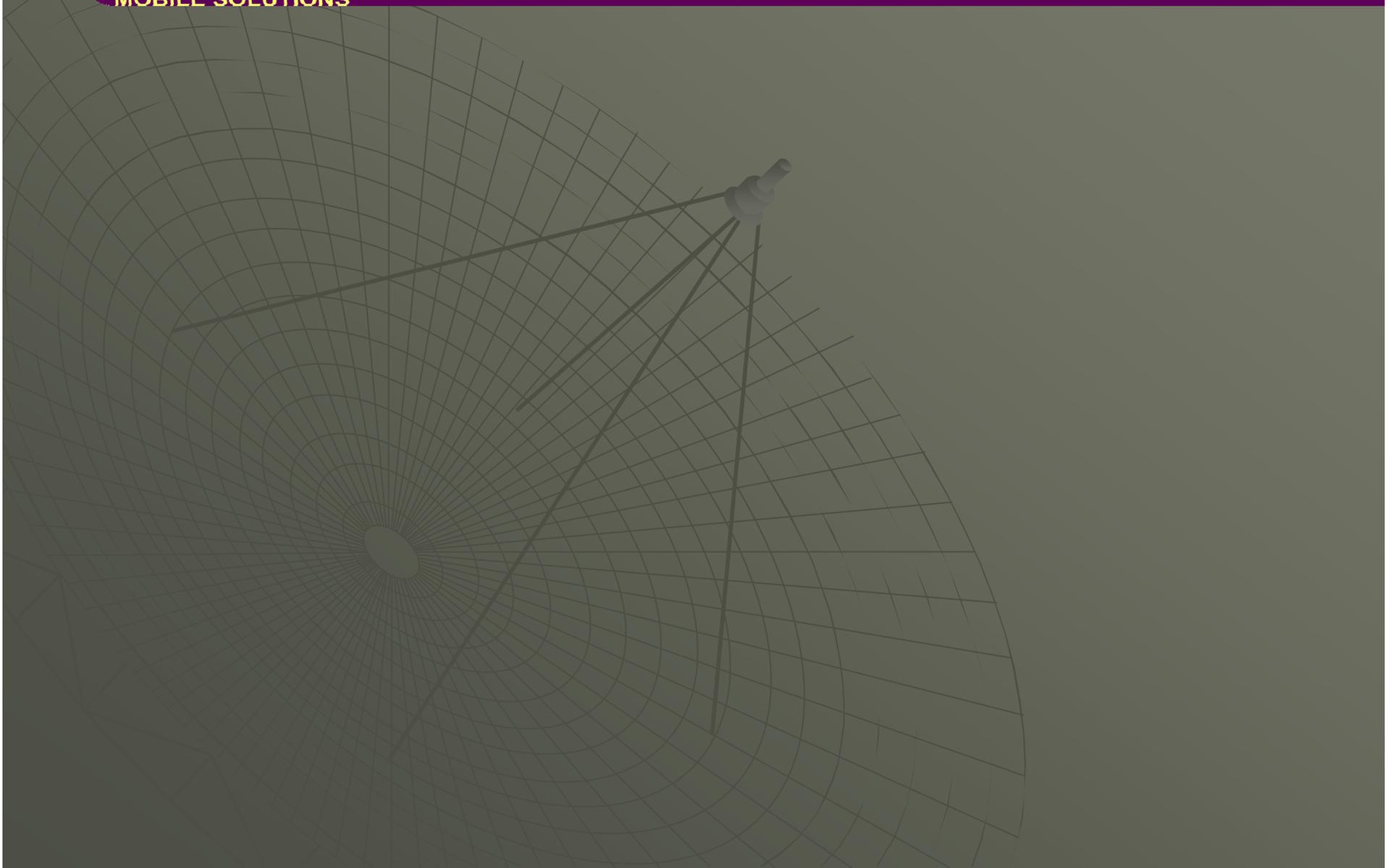
A brief behind the scenes explanation

- At compilation time, the compiler parses all the resource files, extracts the label : data pairs (and structures) and assigns unique numbers to them.
- It creates data tables that relate the labels to the numbers and the numbers to the data.
- It creates a file called R.java which it uses to programmatically assign the numbers to integer constants matching the labels, which is compiled.
- Android uses the constants to retrieve the numbers associated with the labels. As they are static, you can reference them in code.
- Android uses the numbers to access the resource data
- Thus Android can quickly access the resources via the number or the label indirectly via the constant name
- Android can also reference directly via the label, but this is less performant as it requires several parses of files.

- Text resources are defined in xml files in the values directory (or values-xx-rYY etc)
- strings.xml is the default file, but they can be defined in any xml file with the `<resources>` /`</resources>` and `<string >` `</string>` tags
- They are defined as label : value pairs of the form
 - `<string name="/label"/>Text to be associated with label</string>`
- If foreign language or special characters are required, some are supported and can be 'pasted in'. If not or to be safe, use the Java escape character `\` and the Unicode representation such as:
 - `\u00F6` this is ö *Note: will only display if supported by the selected font*
- Individual text resources are referenced by their label
- For locales specific versions, just specify a locale specific value for the same label in a locale specific values directory:
 - Values-en -> strings.xml -> `<string name="hello">Hello</string>`
 - Values-fr -> strings.xml -> `<string name="hello">Bonjour</string>`

- Resources can be automatically referenced in Android Layout files by using the reference of the form '@<resource_type>/<resource_label>'

```
<TextView>  
    android:text="@string/hello"  
</TextView>
```
- All resources can be referenced this way in layouts (assuming it makes sense) and the correct resource is fetched when the layout is inflated at run time.
- The inflater will automatically pick up the resource definition that closest matches the current locale (i.e. via the greatest specificity)



accessing resources programmatically

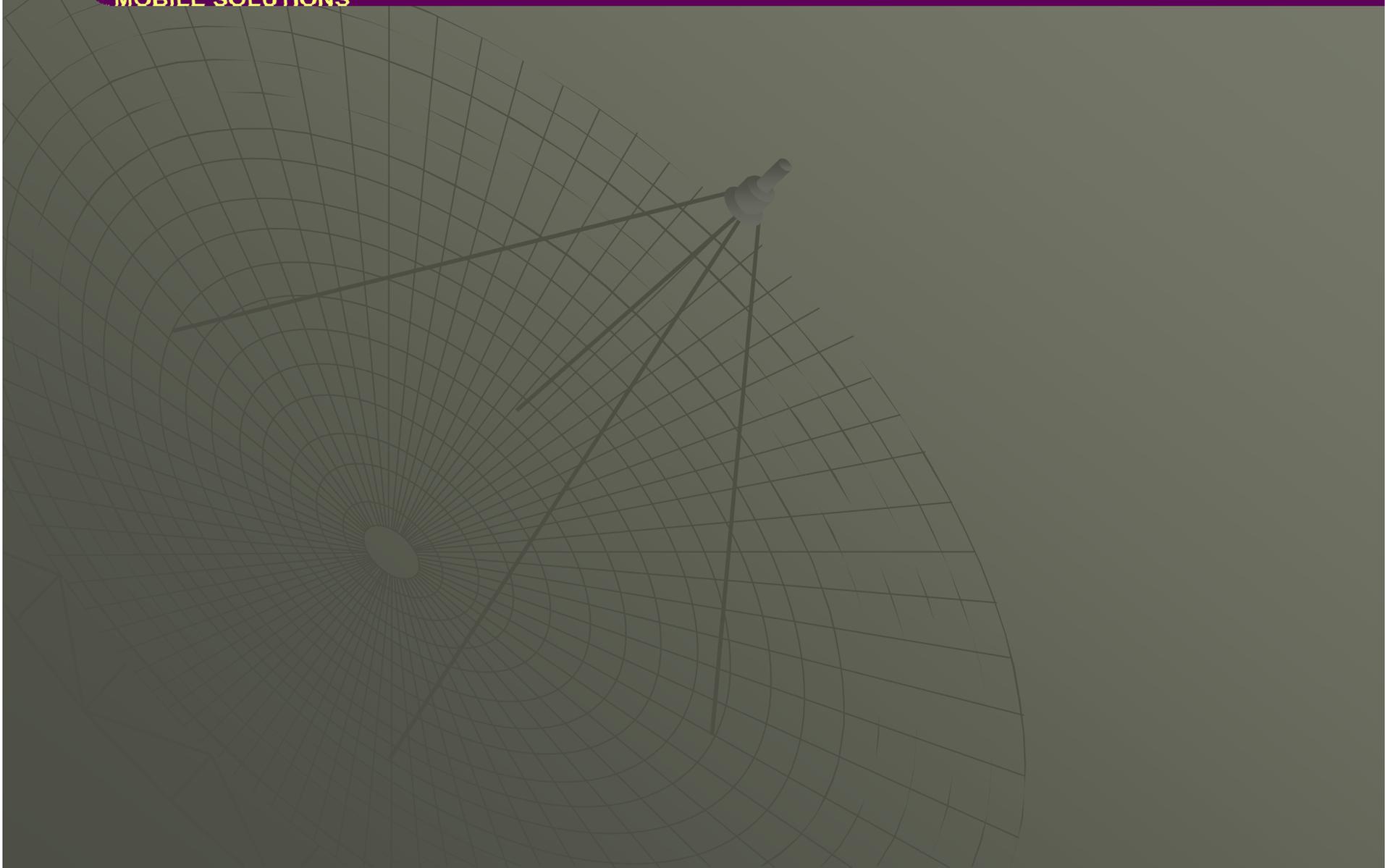
- Most objects that have a property that relates to a resource, can set that property to match a resource programmatically.
- This is usually via a 'setter' method on the class
- Most of these provide as setter method that takes the integer resource reference number.
- These are set by passing the R.<resource_type>.<label> constant

```
TextView textView = (TextView)findViewById(R.Layout.textview1);
textView.setText(R.string.hello);
```
- Occasionally, there is no setter method that takes the integer reference, and the resource must be assigned to its associated object first. This is done via the Resource class. This can be accessed from an Activity like this:

```
String text = getResources().getString(R.string.hello);
```

or if you have the current context like this:

```
String text = context.getResources().getString(R.string.hello);
```



Numbers, dates, calendars are all culturally specific, so need to be able to be changed to match locales. Some methods pick up the currently set Locale, but to make sure you can format them directly:

Number: use the NumberFormat class example:

```
NumberFormat nf = NumberFormat.getInstance(locale);  
String formattedNumber = nf.format(number);
```

Date: use DateFormat class example:

```
Date date = new Date();  
DateFormat df = DateFormat.getDateInstance(DateFormat.style, locale);  
String formattedDate = df.format(date)
```

Time: use DateFormat class example:

```
Date date = new Date();  
DateFormat df = DateFormat.getTimeInstance(DateFormat.style, locale);  
String formattedTime = df.format(date)
```

- There are many different Calendars in use around the world. In the west, we use the Gregorian calendar, and this along with UTC has become the defacto international standard for expressing date and time.
- However, if you want to cater for other cultures, you may well need to use a different Calendar.
- Android provides the Calendar class to define and manage calendars.

- The easiest way to create a Calendar for a specific locale is:

```
Calendar cal = Calendar.getInstance(locale);
```

Which returns a calendar appropriate for the locale containing the current date and time for the default TimeZone.

- Calendars can be manipulated using the methods provided with the Calendar class
- Also see the JodaTime Java library (www.joda.org)

automatically resizing text

One of the 'gotchas' that can come about with multiple languages is that words and phrases have different lengths in different languages.

This can cause a layout to alter as the text contents change – for example, a View widget could increase in width or height, or text could simply be cut off.

In order to avoid having to worry about the size of every single translation, we have developed a View widget that will reduce the size of the text in order to fit it into its parent view.

Will not change the size of the View, but reduces the text to fit into the view given the properties set

Can be included in Layouts and code instead of TextView. Can declare view and attributes in xml layout file (*if attributes included in values > attrs.xml*)

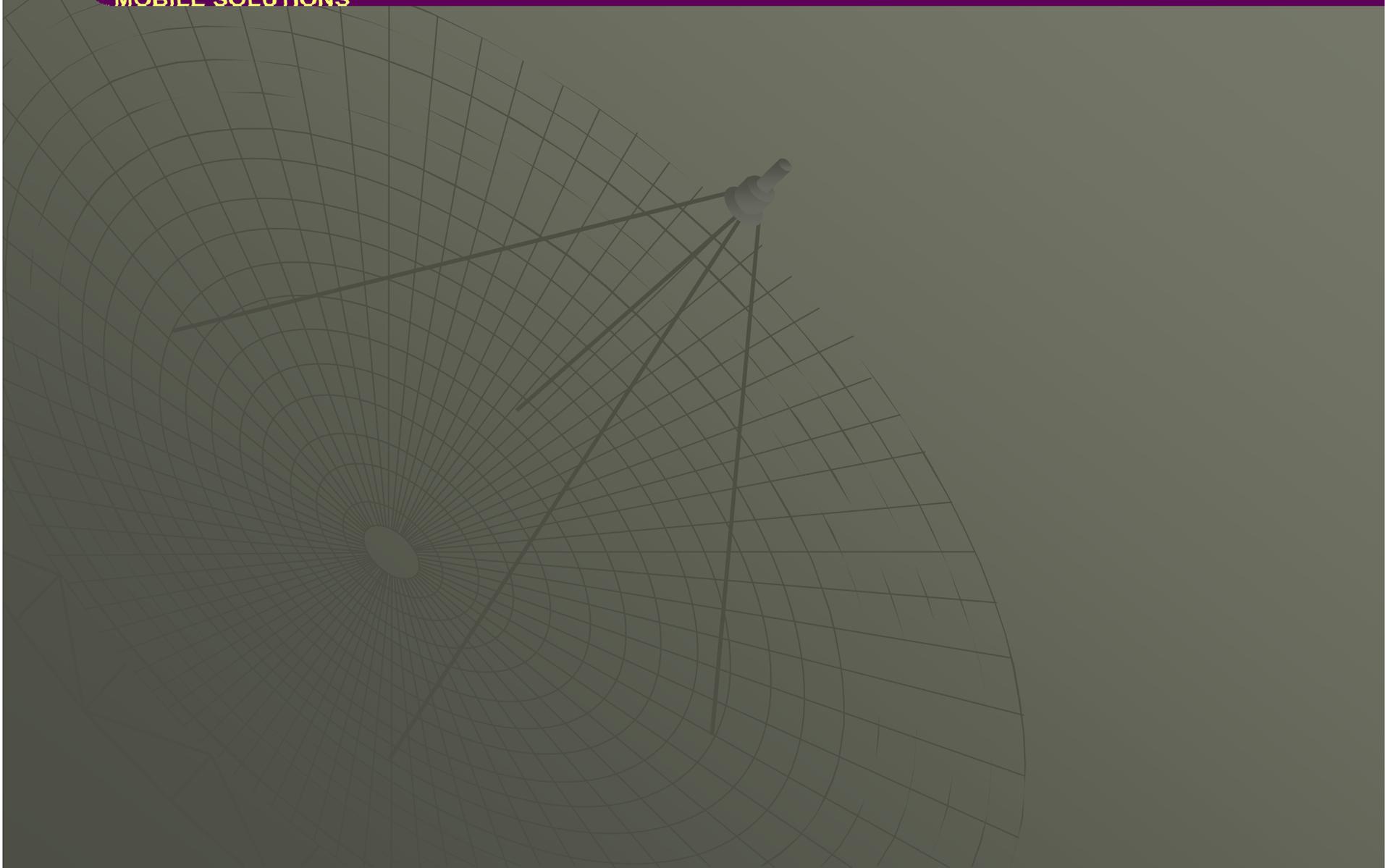
FitLinesTextView

It has the following attributes that can be set:

MinTextPxSize(int)	Sets the min size in Px text reduce to
MinTextSpSize(int)	Sets the min size in Sp text reduce to
MaxTextPxSize(int)	Sets the max size in Px text reduce to
MaxTextSpSize(int)	Sets the max size in Sp text reduce to
MaxLines(int)	Set the max number of lines text can fill
PreferredLines(int)	Sets the no. lines will try to fit to
PreferredSmallerTextPxSize(int)	Sets the preferred smaller text size in Px
PreferredSmallerTextSpSize(int)	Sets the preferred smaller text size in Sp
ReduceIfNotPreferredLines(boolean)	Specifies if it will reduce text size if doesn't fit on preferred number of lines

Inherits from TextView. Requires the class and the associated attribute settings in the attrs.xml file

Source code in the dropbox resource location



This Presentation and Code:

<https://www.dropbox.com/sh/r8j884d8jggwomj/dXUgTvbvyM>

OR

<http://bit.ly/1kxNdiy>



quantum mobile solutions

- QMS is a small, early stage startup that develops its own B2B and B2B2C products and provides consultancy and development contracting services centered around mobile technology
- Our primary product focus is Valens Health Buddy! which improves treatment outcome by increasing prescription adherence through reminders and collecting and analyzing adherence data delivered via Mobile Devices
- We are currently open to Mobile Development consultancy and contracting opportunities and interim development opportunities.

For more information, please contact:

David Allen

Quantum Mobile Solutions

(e): dallen@QMobileSolutions.com

(o): +1 312-376-8669

(c): +1 312-399-2589



questions